Analisis Persoalan *Unlock the Padlock* Pada Google Kickstart 2022 menggunakan *Dynamic Programming* dan *Two Pointer*

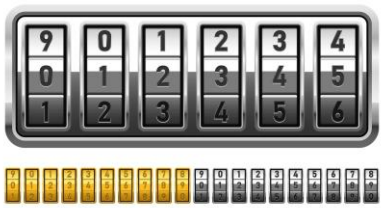
I Gede Arya Raditya Parameswara - 13520036
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13520036@std.stei.itb.ac.id

***Abstract*—Ketika penulis hendak membuka gembok berjenis *combination padlock*, penulis memiliki ide untuk menambahkan sedikit tantangan ketika membukanya. Tantangan yang penulis gunakan untuk membuka gembok ini menggunakan persoalan yang ada pada *Google Kickstart 2022 Round B Problem 3*. Persoalan ini sempat dikerjakan oleh penulis saat kontes berlangsung dan ingin dipraktikan di dunia nyata sehingga penulis membentuk makalah ini untuk analisisnya. Secara garis besar pembukaan gembok ini mirip pada gembok biasanya namun diberikan sedikit aturan-aturan.**

Keywords—dynamic programming; google kickstart; padlock problem; greedy;

I. PENDAHULUAN

Dalam kehidupan sehari-hari ada banyak sekali permasalahan atau persoalan yang bisa ditemukan. Mulai dari membetulkan televisi yang rusak, mencari sepasang kaos kaki dalam tumpukan pakaian, hingga strategi tercepat dalam membuka gembok. Mungkin membuka gembok terdengar hal yang sederhana, namun jika dibumbui dengan aturan-aturan maka perlu algoritma yang efisien untuk membukanya.



Gambar 1.1 Ilustrasi Gembok Jenis Kombinasi *(Sumber: https://www.vecteezy.com/vector-art/1844389-combination-lock-vector-design-illustration-isolated-on-white-background)*

Pada persoalan kali ini, penulis mengutip permasalahan dari Google Kickstart 2022 dalam membentuk aturan-aturan ketika membuka gembok. Google Kickstart merupakan *online coding competition* berdurasi tiga jam yang dirancang oleh engineer Google. Pada Google Kickstart 2022 ronde B, terdapat salah satu soal yang solusinya memerlukan pendekatan pemrograman dinamis. Persoalan ini hanya dapat diselesaikan dengan pemrograman dinamis untuk solusi optimalnya, karena pada Google Kickstart terdapat batasan waktu dan memori dari suatu program untuk berjalan agar jawaban bisa diterima.

Gambar gembok diatas merupakan gembok bertipe *combination padlock*, yang berarti cara membukanya adalah dengan diputar. Berbeda dengan gembok yang menggunakan kunci maupun yang memiliki tombol.

Jenis *combination padlock* ini bisa banyak divariasikan jika ingin dibentuk permasalahan. Seperti banyak langkah untuk menyusun beberapa konfigurasi, mencari langkah terpendek untuk mendapatkan suatu konfigurasi, dan lain-lain.

II. LANDASAN TEORI

A. Google Kickstart

Google Kickstart adalah kompetisi coding yang diselenggarakan oleh Google untuk menemukan orang-orang yang tertarik dengan karir bidang *engineering* di Google dan mengatasi tantangan yang menarik dan sulit. Ini menga dakan kompetisi online sepanjang tahun, memungkinkan pesaing untuk menguji keterampilan peng-*coding*-an mereka. Tidak terbatas pada lulusan mahasiswa ilmu komputer, tetapi menyambut peserta dari segala usia. Google menyelenggarakan tiga turnamen untuk orang-orang dari semua tingkat keahlian. Kickstart adalah salah satunya.

B. Combination Padlock

Kunci kombinasi adalah sejenis alat pengunci yang dibuka dengan memasukkan serangkaian simbol, biasanya angka. Sebuah tombol putar tunggal yang berinteraksi dengan banyak cakram atau kamera, satu set beberapa cakram berputar dengan simbol tertulis yang secara langsung berinteraksi dengan mekanisme penguncian, atau keypad elektronik atau mekanis semuanya dapat digunakan untuk memasukkan urutan. Dari kunci bagasi tiga

digit berbiaya rendah hingga brankas dengan keamanan tinggi, ada sesuatu untuk semua orang. Kunci kombinasi, tidak seperti gembok biasa, tidak memerlukan kunci. Contoh *combination padlock* seperti pada gambar 1.1.

C. *Dynamic Programming*

Algoritma untuk masalah optimasi disebut pemrograman dinamis. Metode pemecahan masalah program dinamis ini melibatkan pemecahan solusi masalah menjadi serangkaian tahap, yang masing-masing dapat dianggap sebagai keputusan yang terpisah. Program dinamis ini tidak ada hubungannya dengan pemrograman atau otomatisasi dengan cara apa pun. Istilah "program" mengacu pada serangkaian instruksi yang diberikan ke komputer untuk menyelesaikan fungsi atau tugas tertentu. Karena menggunakan tabel dan setidaknya ada satu parameter, istilah "dinamis" mengacu pada pencarian solusi yang dapat diubah sesuai kebutuhan.

Program dinamis memecahkan masalah kecil sebelum beralih ke masalah yang lebih besar, atau dari masalah besar yang dipecah menjadi masalah yang lebih kecil. Implementasi perangkat lunak dinamis ini dibagi menjadi dua bagian: top-down dan bottom-up.

Metode top-down adalah metode implementasi program dinamis yang melibatkan penyelesaian solusi secara bertahap. Biasanya dilakukan secara rekursif untuk merekam nilai yang ditemukan sehingga jika fungsi dipanggil kembali dengan argumen yang sama, tidak perlu menghitung ulang untuk mendapatkan penyimpanan sebelumnya.

Pendekatan bottom-up adalah teknik kedua. Metode ini memerlukan penanganan sedikit kesulitan untuk memecahkan masalah yang lebih besar. Teknik ini kadang disebut sebagai teknik pengisian tabel program dinamis karena biasanya dilakukan secara iteratif. Teknik ini telah ditentukan dari kasus dasar yang dapat digunakan dalam perhitungan dalam kasus yang lebih besar ketika mengembangkan formula rekursif.

Items / Weight	0	1	2	3	4	5	6	7	8
1 item	0	0	0	0	5	5	5	5	5
2 items	0	0	5	5	5	5	10	10	10
3 items	0	0	5	5	5	5	10	10	10
4 items	0	0	5	5	5	5	10	10	10
5 items	0	0	5	5	5	8	10	10	10
6 items	0	0	5	5	5	8	10	10	10
7 items	0	0	5	5	8	8	10	11	13
8 items	0	0	5	5	8	8	10	12	13

Tabel 2.1 Ilustrasi Tabel *Knapsack Problem (Sumber: https://readnstart.wordpress.com/2013/07/17/algorithms-solving-the-knapsack-problem-with-dynamic-programming/)*

Knapsack, *coin change*, *longest common subsequence*, dan pemotongan kayu adalah contoh masalah sederhana yang dapat diselesaikan dengan menggunakan algoritma pemrograman dinamis. Masalah knapsack digunakan untuk mengetahui berapa biaya untuk mengambil sesuatu. Pertukaran koin adalah masalah di mana jumlah koin yang optimal diperoleh saat menukar nominal. Tantangan menghitung urutan terpanjang yang mungkin dibuat dari sebuah kata dikenal sebagai urutan umum terpanjang. Biaya optimal untuk memotong sepotong kayu ditentukan oleh masalah pemotongan kayu.

D. *Greedy*

Algoritma greedy adalah setiap algoritma yang menggunakan strategi solusi heuristik untuk menemukan jawaban optimal lokal untuk setiap opsi. Meskipun Algoritma Greedy tidak selalu menghasilkan solusi akhir yang ideal, ia dapat menghasilkan solusi optimal lokal yang dapat mendekati solusi optimal dari suatu masalah dalam waktu yang wajar.

Greedy berarti tamak dan rakus. Orang greedy percaya bahwa kita harus mengambil apa pun yang bisa kita miliki sesegera mungkin. Algoritma greedy menciptakan solusi langkah demi langkah, dan pada setiap tahap, keputusan terbaik harus dibuat dalam memilih pilihan, oleh karena itu tidak ada cara untuk kembali ke langkah sebelumnya dalam algoritma greedy.

Misalnya, Algoritma Greedy memecahkan masalah Travelling Salesman Problem, persoalan yang sangat sulit, dengan mengikuti heuristik "pada setiap tahap dalam perjalanan, kunjungi kota terdekat yang belum pernah dikunjungi." Herustik ini tidak akan membawa kita ke solusi yang ideal, tetapi akan memungkinkan kita untuk mengatasi masalah saat ini dengan lebih cepat.

Algoritma Greedy menghasilkan solusi yang baik untuk beberapa persoalan, tapi banyak pula permasalahan yang tidak memiliki solusi optimal jika menggunakan Greedy. Sebagian besar persoalan yang dapat diselesaikan dengan algoritma greedy harus memenuhi kriteria berikut: *Greedy Choice Property* dan *Optimal Substructure*.

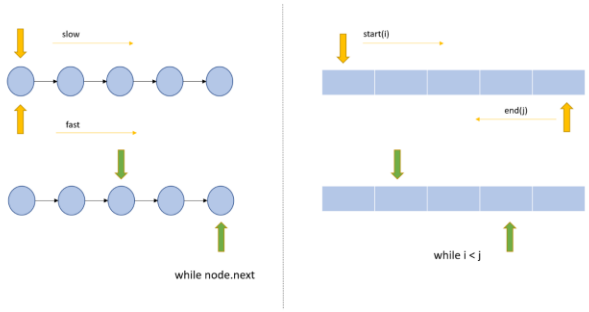
Greedy Choice Property yang berarti Kita dapat memilih pilihan apa pun yang tampaknya terbaik pada saat itu dan kemudian mengatasi submasalah yang muncul. Keputusan algoritma greedy mungkin dipengaruhi oleh pilihan sebelumnya, tetapi tidak oleh pilihan berikutnya atau semua solusi untuk submasalah yang ada. Greedy membuat pilihan greedy demi satu, mengurangi kesulitan saat ini menjadi yang lebih kecil. Atau, dengan kata lain, algoritma greedy tidak pernah mempertanyakan keputusannya. Ini adalah perbedaan utama antara algoritma greedy dan dynamic programming, yang mencari secara mendalam dan menjamin solusi. Setelah setiap tahap, pemrograman dinamis memiliki pilihan berdasarkan semua pilihan yang dibuat pada tahap sebelumnya, dan juga mempertimbangkan kembali jalur algoritma pada tahap sebelumnya untuk sampai pada solusi optimal.

Optimal Substructure yang jika solusi optimal untuk masalah tersebut mengandung solusi ideal lain untuk submasalah tersebut, masalah tersebut dikatakan memiliki substruktur yang optimal. Algoritma greedy tidak dapat memberikan solusi optimal untuk banyak situasi, dan bahkan dapat membuat jawaban unik yang berbahaya. Meskipun demikian, pendekatan ini masih berguna untuk mendekati banyak masalah yang ada, terutama yang membutuhkan waktu pemrosesan yang sangat kompleks dan ekstensif untuk mencapai solusi yang optimal.

E. *Two Pointer Technique*

Algoritma *Two Pointer* adalah pendekatan untuk mengoptimalkan runtime dengan memanfaatkan beberapa urutan (tidak harus mengurut) data. Ini umumnya diterapkan pada daftar (array) dan *Linked List*. Ini umumnya digunakan untuk mencari pasangan dalam array yang diurutkan. Pendekatan ini bekerja dalam ruang konstan.

Tempat untuk memulai. Bergantung pada apa yang akan lakukan, pointer dapat ditempatkan di mana-mana. Kedua pointer di bagian kiri gambar dimulai di lokasi yang sama, yaitu awal daftar tertaut. Sedangkan pada bagian kanan gambar, pointer terletak di kedua ekstrem satu di indeks awal dan yang lainnya di indeks akhir.



Gambar 2.2 Ilustrasi *Two Pointer (Sumber: https://towardsdatascience.com/two-pointer-approach-python-code-f3986b602640)*

Cara konvergen menuju solusi akan ditentukan oleh pergerakan pointer. Pointer mungkin berjalan dengan cara yang sama atau ke arah yang berlawanan (bagian kanan gambar). Namun juga memiliki alternatif kenaikan untuk kedua pointer (bagian kiri gambar).

Kondisi stop menentukan kapan kita harus berhenti. Dengan cara melanjutkan di bagian kiri sampai mencapai simpul yang elemen berikutnya adalah tidak ada atau hingga pointer kanan bernilai lebih kecil dari yang kiri.

III. METODOLOGI DAN ANALISIS

Pada bagian ini, akan dipaparkan permasalahan dari "Unlock the Padlock" serta dua metode penyelesaian, yaitu Greedy dan Dynamic Programming, serta menganalisis perbandingan kompleksitas dari kedua metode tersebut mulai dari kompleksitas waktu serta kompleksitas memori. Selain pemeriksaan kompleksitas, analisis juga akan dilakukan pada hasil jawaban yang lebih optimal.

A. *Deskripsi Permasalahan*

Bayangkan Anda memiliki gembok, yang merupakan kunci kombinasi yang terdiri dari N dial, yang awalnya disetel ke kombinasi acak. Dial gembok berukuran D, yang berarti dapat memiliki nilai antara 0 dan D − 1, inklusif, dan dapat diputar ke atas atau ke bawah. Mereka juga diurutkan dari kiri ke kanan, dengan dial paling kiri dan paling kanan di posisi 1 dan N, masing-masing. Gembok dapat dibuka dengan mengatur nilai semua tombolnya ke 0.

Anda dapat melakukan nol atau lebih operasi semacam ini:

- Pilih rentang [l, r] apa pun sehingga $1 \le l \le r \le N$ dan putar semua tombol di [l, r] bersama-sama, ke atas atau ke bawah. Memutar ke atas meningkatkan nilai setiap dial dalam rentang [l, r] sebesar 1, dan memutar ke bawah mengurangi nilainya sebesar 1. Perhatikan bahwa dial dengan nilai D − 1 menjadi 0 saat dinaikkan (diputar ke atas) dan dial dengan nilai 0 menjadi D − 1 saat diturunkan (diputar ke bawah).

Serangkaian operasi harus memenuhi kondisi berikut:

- Rentang $[l_i - 1, r_i - 1]$ yang dipilih dalam operasi $(i - 1)$-th harus benar-benar terkandung dalam rentang $[l_i, r_i]$ yang dipilih dalam operasi ke-i; yaitu, $l_i \le l_i - 1 \le r_i - 1 \le r_i$. Rentang awal ($[l_1, r_1]$) dapat dipilih secara sewenang-wenang.

Contoh urutan operasi yang valid untuk membuka gembok dengan kombinasi awal [1, 1, 2, 2, 3, 3]:

1. Putar rentang [5, 6] ke bawah.
2. Putar rentang [3, 6] ke bawah.
3. Putar rentang [1, 6] ke bawah.

Berikut ini adalah beberapa operasi yang tidak dapat dilakukan:

1. Rentang putaran [1, 4] setelah [6, 9], karena [6, 9] tidak sepenuhnya terdapat dalam [1, 4] (tidak memenuhi $r_i - 1 \le r_i$ di mana $r_i - 1 = 9$ dan $r_i = 4$) .
2. Rentang putaran [3, 6] setelah [2, 7].

Tujuannya bagi Anda adalah untuk menghasilkan jumlah minimum operasi valid yang diperlukan untuk membuat semua panggilan di gembok disetel ke 0.

B. *Metode Pemrograman Dinamis*

Pada penyelesaian masalah menggunakan metode pemrograman dinamis, penulis mengimplementasikan metode bottom-up dynamic programming. Penulis membentuk *array* tiga dimensi bernama dp yang bermakna, untuk dp[i][j][0] berarti solusi optimal untuk mengubah seluruh angka pada gembok menjadi bernilai angka ke-i. Sedangkan untuk dp[i][j][1] berarti solusi optimal untuk mengubah seluruh angka pada gembok menjadi bernilai angka ke-j. Dengan i dan j adalah dua pointer dari gembok.

Di awal program, penulis menginisialisasi untuk setiap dp[i][i][0] dan dp[i][i][1] bernilai 0. Agar ketika melakukan program *dynamic programming* utama sudah memiliki nilai awal untuk mendapatkan nilai-nilai optimal lainnya. Nilai inisialisasi tersebut didapat karena merupakan kasus unik, jelask bahwa pada urutan angka gembok ke i dan i pastilah jumlah putaran yang perlu dilakukan bernilai 0.

Dalam mencari nilai optimal lainnya, program akan mengecek nilai minimum jika membentuk semua angka sepanjang i sampai j bernilai angka ke-i dengan membentuk semua angka bernilai angka ke-j.

Berikut adalah potongan kode yang telah dibentuk oleh penulis pada kontes Google Kickstart 2022 round B.

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define fi first
#define se second
```

```
#define pb(a) push_back(a)
#define mp(a, b) make_pair(a, b)
#define pii pair<int,int>
#define vec vector
#define el '\n'

int fmin(ll x, ll y, ll d){
  if(y < x) swap(x, y);
  return min(y-x, d+x-y);
}

int main(){
  int tc; cin >> tc;
  for(int Case = 1; Case <= tc; Case ++){
    ll n, d;
    cin >> n >> d;

    vector<ll> v(n);
    ll dp[n+5][n+5][2], ans;

    for(int i = 0; i < n; i++){
      cin >> v[i];
      dp[i][i][0] = 0;
      dp[i][i][1] = 0;
    }

    for(int len = 2; len <= n; len++){
      for(int i = 0; i+len-1 < n; i++){
        int l = i, r = i+len-1;

        dp[l][r][0] = min(
          dp[l+1][r][0] + fmin(v[l],v[l+1],d),
          dp[l+1][r][1] + fmin(v[l],v[r],d)
        );

        dp[l][r][1] = min(
          dp[l][r-1][1] + fmin(v[r],v[r-1],d),
          dp[l][r-1][0] + fmin(v[r],v[l],d)
        );
      }
    }

    ans = min(
      dp[0][n-1][0] + fmin(0,v[0],d),
      dp[0][n-1][1] + fmin(0,v[n-1],d)
    );

    cout << "Case #" << Case << ": " << ans << el;
  }
}
```

Potongan Kode 3.1 Solusi Pemrograman Dinamis "Unlock the Padlock" *(Sumber: Dokumen Penulis)*

Pada kode tersebut, penulis membentuk fungsi bernama fmin(x, y, d) yang berfungsi untuk menghitung banyaknya putaran yang harus dilakukan jika hendal mengubah dari angka x ke angka y jika banyaknya angka pada satu putaran ada sebanyak d.

Di akhir program, penulis menghitung banyaknya putaran yang harus dilakukan untuk mengubah semua angka pada gembok menjadi 0. Setelah didapat jawabannya, program akan mengeluarkan output sesuai ketentuan Google Kickstart 2022.

Setelah program selesai, penulis melakukan tes terhadap salah satu *testcase* yang tersedia, seperti pada gambar berikut ini,

Gambar 3.1 *Test Case* "Unlock the Padlock" *(Sumber: https://codingcompetitions.withgoogle.com/kickstart/round/00000000008caa74/0000000000acef55)*

Dan ketika dijalankan, program memberikan output seperti berikut,

Gambar 3.2 *Output* Solusi Penulis *(Sumber: https://codingcompetitions.withgoogle.com/kickstart/round/00000000008caa74/0000000000acef55)*

Setelah selesai menjalankan *testcase*, penulis melakukan submit untuk mendapatkan *verdict* dari solusi penulis secara keseluruhan, karena testcase hanya berguna untuk mengecek apakah kode program saat ini sudah dapat berjalan untuk kasus-kasus tertentu atau tidak. Ketika submisi dilakukan, terdapat tiga jenis test set. Pertama, nilai $D = 2$ dan $1 \leq N \leq 40$, kedua, $2 \leq D \leq 10$ dan $1 \leq N \leq 40$, ketiga, $2 \leq D \leq 10^9$ dan $1 \leq N \leq 400$.

Gambar 3.3 *Verdict* Solusi Penulis *(Sumber: https://codingcompetitions.withgoogle.com/kickstart/round/00000000008caa74/0000000000acef55)*

Ternyata solusi pemrograman dinamis ini memenuhi seluruh *Test Set* yang di bentuk oleh Google. Sehingga terbukti bahwa solusi ini optimal, cepat, dan berhasil menyelesaikan masalah.

C. Metode Greedy

Jika diperhatikan, pada permasalahan ini, urutan memutar gembok bisa dimulai dari ujung kiri ke ujung kanan terlebih dahulu, lalu langkah selanjutnya mengecil, mengecil, dan mengecil hingga semuanya selesai diputar dan berangka 0.

Dalam penyelesaian menggunakan metode greedy ini digunakan pula teknik two pointer dalam menyelesaikannya dengan cara melakukan pemilihan solusi lokal terbaik setiap langkah ketika memutari gembok. Sebagai contoh jika ujung paling kiri bernilai 2 dan ujung paling kanan bernilai 3, dengan jumlah putaran ada 9. Maka akan diambil 2 putaran kebawah sebagai langkah greedy. Lalu akan melanjutkan langkah ini lagi dari ujung kiri kedua dan ujung paling kanan, dan seterusnya hingga semuanya bernilai 0.

Berikut adalah potongan kode dari metode greedy dengan two pointer,

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define fi first
#define se second
#define pb(a) push_back(a)
#define mp(a, b) make_pair(a, b)
#define pii pair<int,int>
#define vec vector
#define el '\n'

int fmin(ll x, ll y, ll d){
  if(y < x) swap(x,y);
  return min(y-x, d+x-y);
}

int main(){
  int tc; cin >> tc;
  for(int Case = 1; Case <= tc; Case ++){
    ll n, d, ans, diff, l, r;
    cin >> n >> d;

     l = 0;
     r = n-1;
     vector<ll> v(n);
     for(int i=0; i<n; i++) cin >> v[i];

     ans = 0;
     diff = 0;
     while(l <= r){
       int lval, rval, le, ri;
       le = (v[l] + diff) % d;
       ri = (v[r] + diff) % d;

       lval = fmin(0, le, d);
       rval = fmin(0, ri, d);

       if(lval <= rval) {
         ans += lval;
         diff += (lval == le) ? -lval : lval;
         l++;
       } else {
         ans += rval;
         diff += (rval == ri) ? -rval : rval;
         r--;
       }

       diff = (diff + d) % d;
     }

     cout << "Case #" << Case << ": " << ans << el;
   }
}
```

Potongan Kode 3.1 Solusi Greedy "Unlock the Padlock" *(Sumber: Dokumen Penulis)*

Setelah program menggunakan metode greedy tersebut dijalanken menggunakan testcase yang sama dengan sebelumnya (gambar 3.2), didapatkan hasil yang sama pula seperti gambar 3.3. Berarti metode siap untuk disubmit untuk pengecekan menyeluruh.

Gambar 3.3 *Output* Solusi Penulis dengan Metode Greedy *(Sumber: https://codingcompetitions.withgoogle.com/kickstart/round/00000000008caa74/0000000000acef55)*

Namun, ternyata solusi ini tidak memberikan solusi paling minimum pada test set 2 ($2 \leq D \leq 10$ dan $1 \leq N \leq 40$). Berarti solusi ini tidak seefektif dan seoptimal metode pemrograman dinamis.

D. Analisis Kompleksitas Kedua Metode

Ketika menganalisis kedua metode ini, penulis membandingkan dua jenis kompleksitas yang paling krusial, yaitu waktu dan memori. Karena kedua kompleksitas ini menjadi kriteria keberhasilan suatu jawaban pada Google Kickstart 2022.

Batas waktu yang diperbolehkan dalam kontes Google Kickstart ini adalah 30 detik dan batas memorinya 1 GB. Berikut adalah kompleksitas kedua metode.

Metode	Jenis Perbandingan	
	Time Complexity	***Memory Complexity***
Dynamic Programming	$O(n^2)$	$O(n^2)$
Greedy	O(n)	O(n)

Tabel 3.1 Tabel Perbandingan Kompleksitas Kedua Metode
(Sumber: Dokumen Penulis)

Jika diperhatikan, metode greedy unggul lebih jauh dibandingkan dengan metode pemrograman dinamis, dengan kompleksitas waktu dan memorinya yang linear. Namun, walaupun pemrograman dinamis memiliki kompleksitas $O(n^2)$ solusi ini tetap diterima oleh Google Kickstart, karena waktu jalan programnya tidak melebihi 30 detik dan kurang dari 1 GB, mengingat nilai N maksimal hanyalah 400.

Namun, penggunaan metode greedy dengan teknik two pointer tidak dapat menyelesaikan seluruh test yang disediakan oleh Google. Sehingga untuk uji keoptimalan, tetap pemrograman dinamis yang unggul karena dapat menyelesaikan semua test set dengan benar.

IV. KESIMPULAN

Pada analisis di bab sebelumnya, penulis menarik kesimpulan bahwa metode yang paling optimal untuk menyelesaikan permasalahan "Unlock the Padock" ini adalah menggunakan Pemrograman Dinamis.

Penggunaan metode Greedy dengan Two Pointer Technique memang dapat menyelesaikan permasalahan tersebut namun hanya untuk beberapa kasus saja, karena kasus sisany tidak memiliki jawaban yang paling optimal. Hal ini disebabkan karena permasalahan tidak memenuhi kriteria *Greedy Choice Property* dan *Optimal Substructure*.

Hal ini membuktikan bahwa greedy tidak selalu mendapatkan hasil yang optimal, sedangkan pemrograman dinamis pasti mendapatkan hasil yang optimal karena pemrograman dinamis mencari solusi terbaik dari kasus terkecil lalu dilanjutkan ke kasus yang besar.

Penggunaan pemrograman dinamis pada solusi penulis memiliki teknik bottom-up yang berarti memulai dari kasus-kasus yang kecil dan unik, lalu perlahan membesar ke kasujs yang besar. Penggunaan teknik ini karena untuk mencari jawaban dengan kasus besar memerlukan jawaban kasus yang lebih kecil terlebih dahulu.

Oleh karena itu, solusi paling efektif untuk permasalahan ini adalah menggunakan teknik *Bottom-Up Dynamic Programming*.

ACKNOWLEDGMENT

Penulis memanjatkan rasa syukur kepada Tuhan yang Maha Esa karena atas rahmat-Nya sehingga makalah ini dapat diselesaikan dengan lancer dan tepat waktu. Penulis juga ingin mengucapkan terima kasih yang sebesar-besarnya atas bimbingan dan ilmu yang diberikan oleh Dr. Ir. Rinaldi, M.T. selaku dosen K03 mata kuliah IF2211 Strategi Algoritma 2021/2022.

REFERENCES

[1] Bloomfield, Aaron; Sotomayor, Borja. "A Programming Contest Strategy Guide" (PDF). SIGCSE '16: Proceedings of the 47th ACM Technical Symposium on Computing Science Education.

[2] Munir, Rinaldi. 2022. Slide Kuliah IF2211 Strategi Algoritma, Bandung

[3] Gozali, W., & Aji, A. F. (n.d.). Pemrograman Kompetitif Dasar (1.1st ed.).

[4] Anany,Levitin. Introduction to the Design & Analysis of Algorithms, 3rd Edition, Addison-Wesley, 2012

[5] Google Kickstart 2022, Round B, Problem 3, "Unlock the Padlock", 24 April 2022.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2022

I Gede Arya Raditya Parameswara
13520036